OS 444 Group 36

May 27, 2018

Project 3: The Kernel Crypto API

Group 36

Scott Russell
Arya Asgari
Fischer Jemison

Contents

1 Design Plan

For our implementation of this assignment we used a simple version of SBD block driver. In our ReadMe we have linked to all the appropriate sources for our code. All the code used was provided open source and was recommended that we DO NOT create our own block device driver as it would be a much larger scale than the scope of this assignment or the class. The main purpose of this assignment is to get us more familiar with block devices and how to work with modules in the kernel.

Block Devices are used to host a file system. Usually they only handle I/O operations that transfer 1 or more blocks of data. These blocks of data are of size 512 bytes. Because of how Linux works the ability to read and write can be used as if it was a char device (any number of bytes at the same time). Block devices can be accessed through a file system node.

The Crytpo API is able to encrypt stored messaged in SBD. We use it to be able to encrypt and decrypt the files we access on read and write. In the specifications we are told to "... set the key as a module parameter." Crypto code was modified and taken from another source listed in the readme file. Meaning that we have to access the key as a module which is where the use of modules comes in for this assignment. Otherwise it would be simple to ignore modularity and run the key locally. However for this assignment we are working with the kernel module. Giving us more experience in this field.

2 Version Control: Table

Commit Hash	Commit Description	Date Added
288fc96	Add patch file	Sat May 26th 2018
2f728b6	Updates readme with full instructions	Sat May 26th 2018
a350d57	Updates makefile	Sat May 26th 2018
05ca416	Updates sdb to compile	Thu May 24th 2018
ead0c9c	Update ReadMe.md	Tue May 22nd 2018
62cb5cb	Update ReadMe.md	Tue May 22nd 2018
7eaffc7	Update ReadMe.md	Tue May 22nd 2018
d0e9656	Delete readme	Tue May 22nd 2018
d741dac	Add files via upload	Tue May 22nd 2018
76585c8	Rename sdb.c to hw3/sdb.c	Tue May 22nd 2018
72bb0d1	Rename hw3/updated.c to sdb.c	Tue May 22nd 2018
982e4c2	updated with crypto	Tue May 22nd 2018
0d494ed	Create cryptoloop-original.c	Wed May 17th 2018
0833065	Copied over C code From Source	Wed May 17th 2018
f6f22ab	Added readme with Helpful Resources Linked	Wed May 17th 2018

3 Work Log: What was done when?

May 16th	Research and Latex creation
Scott Russell	Did some research into Open source Crypto Linux API to use
Arya Asgari	Made overleaf template and researched module creation
Fischer Jemison	Block Driver Research
May 17th	**Initial Setup to GitHub**
Scott Russell	Pushed initial crypto API and Block Driver Template to Github
May 23rd	**Bulk of Kernel Debugging and Latex work**
Scott Russell	Helped with Latex writeup (Log of Commands and Design Plan)
Fischer Jemison	Worked on debugging Crypto + Block Driver
Arya Asgari	Latex Writeup (Assignment Questions) and debugging

4 Assignment Questions

4.1 Main Point of Assignment

It seems that the main point of this assignment was to gain a better understanding of block drivers and the Linux kernel's crypto API. It was recommended that we not create our own block device driver and instead use one that was already made. Therefore, the focus shifted towards becoming familiar enough with the block driver we chose to add encryption to it. In order to do so, we also had to become familiar with the Linux kernel's crypto API. This was all to be done as a kernel module, so we needed to become familiar with building and running those as well.

4.2 Approach to Problem

This entire assignment was an experiment for us to get more familiar with and utilize block drivers and crypto API that was previously implemented. We had to research and have an understanding of these functionalities in order to combine them together into our implementation. Since we did not have a strong understanding of the code initially this assignment was useful in forcing us to experiment with different elements of the code to achieve the goal of the crypto API combined with the shell of the block driver.

4.3 Ensuring Correctness

At the bottom of the ReadMe file we discuss the commands used in creation and utilization of the kernel module functionality. Needless to say allot of the difficulties we had with our project was trying to get the code to work on the kernel server after testing locally. It is very difficult to be able to understand errors on a kernel level and working with the kernel hardware is fickle to say the least. We tested our code against being able to successful write and read from the disk. We could also check the encryption and decryption here as we can see if the block device read/write is working properly. being able to grep for a string the a newly mounted test tested for additional functionality.

4.4 What We Learned

Unlike most assignments in programming we did not have to create our solution entirely from scratch. This is very common in industry where we have to deal with old, undocumented, and sometimes poorly written software. Then you are given to task to understand, recreate, or combine the code. Talking with mentors in CS they emphasized that writing code from scratch is extremely rare for fresh graduates. Usually you are working with pre-existing code and teams of programmers. This assignment was a great real world example of utilizing pre-existing code to create a new solution.

5 Log of Commands

```
Links to Resources Used (Template for Block Driver and Crypto Stuff)
 Discussion of Linux Device Drivers: https://lwn.net/Kernel/LDD3/
Link to Block Driver Source: (Used in project)
 ↪ http://blog.superpat.com/2010/05/04/a-simple-block-driver-for-linux-kernel-2-6-31/
This link also includes: the Makefile and Module examples to compare with CryptoLoop
 ↪ modules
This above Code was combined with the CrytoLoop enabling

Module Found here:
 ↪ https://elixir.bootlin.com/linux/v3.14.26/source/drivers/block/cryptoloop.c
%SOURCE FOR KERNEL MODULE COMMANDS: https://wiki.archlinux.org/index.php/Kernel_module

How to build and test block driver:

In order to create Multiple Sessions to run concurrently software like tmux can be used

Apply our patch file by running patch -p0 -i hw3.patch from the base of the linux-yocto
 ↪ directory.

Run make menuconfig

Select Device Drivers

Select Block devices

Select SBD and press the spacebar. An "M" should appear between the angle brackets next
 ↪ to the name.

Save and exit the configuration
```

Run make -j4 all to build the kernel and all modules. You may see a warning from sbd.c but it should compile successfully.

Make sure you have a second terminal session open. We recommond using tmux new and **then** <CTRL+B><%> to open two vertical terminal sessions.

In each terminal session, source the file that will configure bash settings to utilize the qemu environment with source /scratch/opt/environment-setup-i586-poky-linux

Start the vm with qemu-system-i386 -gdb tcp::5622 -S -nographic -kernel linux-yocto-3.19/arch/x86/boot/bzImage -drive file=core-image-lsb-sdk-qemux86.ext3 -enable-kvm -usb -localtime -net user,hostwd=tcp::55360-:22 -net nic --no-reboot --append "root=/dev/hda rwconsole=ttyS0 debug". This will start with networking enabled and forward host port 55360 to port 22 on the virtual machine, allowing you to connect ssh and scp via port 55360.

Enter root as the username.

On the second terminal, from the base linux-yocto directory, run scp -P 55360 drivers/block/sbd.ko root@localhost:/home/root to transfer the kernel module to the virtual machine.

In the terminal hosting the vm, run ls in the base directory **for** the root user. You should have a file called sbd.ko.

to load the module: insmod sbd.ko

lsblk will show that it is loaded

mkfs -t ext2 /dev/sbd0 creates a file system **for** the module

mkdir /mtn creates a new folder location **for** interaction with the module

mount -t ext2 /dev/sbd0 /mtn mount the module in /mtn with the ext2 file system

lsblk -f should show the mounted module

echo "sample string" > /mtn/**$(**folder name**)** writes to the module cat /mtn/**$(**folder name**)** reads from the module

A grep **for** the sample string should **return** nothing **if** it is encrypted properly

When you are **done** with the module, unmount it with umount /mtn

Remove the module with rmmod sbd.ko